Description of an Architecture for Critical Distributed and Real-Time Data Collection Applied to MVEDR

Fernando Consigli[1], Enrique Gallego[1], Hugo Ramón[1], Horacio Villagarcía Wanza[1, 2]

III LIDI – Facultad de Informática – UNLP[1]
Calle 50 y 120 – 2do piso – (1900) La Plata, Argentina

CIC – Comisión de Investigaciones Científicas Pcia. Bs As.[2]
526 entre 10 y 11 – (1900) La Plata, Argentina

consigli@argentina.com, kike.unlp@gmail.com
{hramon, hvw}@info.unlp.edu.ar

Abstract. Nowadays, motor vehicle safety has gained a widespread interest. Technological breakthroughs have allowed more efficient system implementations as regards life protection elements for both passengers and pedestrians. Among them, event data recorders (EDRs) can be found.
In this paper, we introduce an architecture and hardware simulator to satisfy the Institute of Electrical and Electronics Engineers (IEEE) 1616 standard as well as the Argentinean traffic laws (law 24.449 art. 29 and law 26.363). In order to accomplish this, we built an architecture using Transaction Level Modeling (TLM) methodology, accompanied by a hardware description language called SystemC, proving the vantages of using a high level approach for modeling real time embedded systems.

Keywords: data collection, distributed real-time, TLM, MVEDR, co-design, SystemC

1 Introduction

In order to obtain crash event data, it is necessary to have a mechanism that registers car variables in real time as the accident takes place. This goal is achieved by using a Motor Vehicle Event Data Recorder (MVEDR), referred as black box. This device is in charge of storing information that can be employed to identify accident causes and rebuilt to analyze it [1].

In the United States of America, approximately 80% of all passenger vehicles since 2004 are equipped with MVEDRs.

The Argentinean automobile safety law states that all cars sold in Argentina must include a safety device capable of recording moments surrounding the event. By doing so, it ought to enable technicians to analyze the actual causes of accidents and take preventive actions to improve road safety.

In 2004, IEEE published an international standardization of motor vehicle event data recorders entitled "Standard for Motor Vehicle Event Data Recorders (MVEDR)" [2]. This document specifies a protocol for MVEDR output data compatibility and export protocols of MVEDR data elements.

It would be useful to adopt, from now on, techniques that allow the development of devices that complies with the aforementioned regulations.

Simulations are a powerful tool during development because they help designers and engineers to make high-costs design decisions as they are able to foresee how the system will react to certain situations.

Despite the fact that simulating a system is not the same as producing a real model, it is a notable enhancement in development processes.

In this particular case, simulations may let us find out which would be the optimum architectural design that leads to a real implementation observing the IEEE standard without the need of actually implementing it.

Nevertheless, if we aim to employ simulation techniques, we need to utilize the appropriate methodology.

We organize the paper as follows, in section 2 we despite architecture design technology used, section 3 highlights modeling constraints defined by IEEE 1616 std and domain application, section 4 shows proposed architecture, section 5 describe the results and section 6 provides some concluding remarks.

2 MVEDR architecture design

In the past, software and hardware designers lacked a standardized mechanism to communicate requirements between tasks in the development process. System specifications and functionality were defined through a System Architectural Model (SAM) [3], which did not permit to have an early executable model. Consequently, it makes it difficult to introduce changes in requirements, either functional or non-functional.

In order to overcome this obstacle, Transaction Level Modeling [4] was presented thus enabling a new methodology.

2.1 TLM

TLM is based on the concept of modeling only the level of detail that is needed by the teams developing the system components and subsystem for a particular task.

Using TLM along with a hardware description language, like SystemC [5], permits the use of an early executable system model, improving design tasks and easing testing jobs with the utilization of simulations, without the need of a physical hardware implementation.

One of the main characteristics of TLM is the disaggregation between functional components and communication, which allows independent refinement of components.

Basically, we can distinguish three levels of refinement. *Un-timed*: only functionality is represented. A model with un-timed communication and functionality is usually referred to as system architectural model. *Approximately-timed*: some basic timing information is included. The real time duration is approximated. *Cycle-timed*: the details of implementation enable a cycle-accurate simulation.

Starting from a SAM, considering that is un-timed both in communication as in functionality, it can be derived into a TLM with un-timed communication and approximately-timed functionality. Subsequently, components are refined separately until they reach a cycle-timed level (extremely precise hardware approach).

A model that is cycle-timed shows accuracy of communication and of functionality as well. It is usually referred to as Register-Transfer Level (RTL) model [6]. This type of model often requires large amounts of time to simulate.

Communication in TLM is modeled by means of channels. Functional components communicate between them with transactions, which are carried out using interface functions implemented by channels.

This methodology allows a larger reuse of design components, not only within a certain project, but also on future projects, hiding finer implementation details.

2.2 SystemC

SystemC [7][8][9] is a system description language whose aim is to provide, both to designers and architects, with a standard based on C++ for development of software and hardware hybrid systems.

SystemC is a library that provides the necessary constructors for hardware modeling that enable concepts of time, hardware data types, hierarchy and structure, communications and concurrency.

The different elements that make up a system are: *Sc_module*: it is the smaller container of functionality with state, behavior and structure for hierarchical connectivity. It may represent software, hardware or any physical entity. It is represented with rectangles. *Sc_channel*: it is the mechanism that models communication. It can represent either simple medium like a wire or a first-in-first-out (FIFO) data structure, or a complex scheme like a Controller Area Network (CAN) bus. It implements the methods required to model the behavior of the channel. It can be represented with a hexagonal shape. *Sc_port*: It is the element within a module that is bound to a channel, so as to realize the connection between them. It is represented by a square with directional arrows that indicate the primary flow of information. *Sc_interface*: it provides the virtual declaration of methods implemented by Sc_ports and Sc_channels. *Sc_thread* and *Sc_method*: these are the basic execution units. They are contained inside a module. Sc_threads are started only once, and they can be suspended through “wait” sentences. On the contrary, Sc_methods are executed multiple times and cannot be suspended (as a matter of fact, time does not pass between a call and its return). They are represented with circles. *Sc_time*: it is used to measure time and is expressed in two parts: a numeric magnitude and a time unit. *Sc_event*: it represents an event, which is defined as something that happens at a specific point in time. Sc_methods and Sc_threads flow of execution may be

determined by the occurrence of a specific Sc_event. It is represented with an arrow. *SystemC* predefined data types: a set of arithmetic, boolean and fixed-point data types that are provided in order to support different hardware representations.

Lastly, SystemC provides a simulation kernel that controls the flow of simulations so that the model behaves as it was described. This kernel is in charge of advancing time and scheduling the execution of simulation processes.

3 Architecture modeling constraints

System-on-chip devices have numerous constraints regarding physical and logical limitations. These constraints are related to IEEE 1616 std and domain application.

3.1 IEEE 1616 Constraints

IEEE standard does not determine specific technology to make a MVEDR. Yet, it mentions the following likely primary components: *Processor*: for managing or assisting data organization, retrieval, retention, or delivery of collected data. *Non-volatile Memory*: for retaining key data that occurred immediately before and after an event. It should not require a power source to retain its contents. *Memory Buffer*: for collecting data just prior to an event which can be refreshed, and to dump its content into a non-volatile memory shortly after an “event” or “crash”. *Internal or External Clock*: for keeping relative or real time when an event occurs. Furthermore, it determines when each data element happened in relation with the event.

3.2 Memory constraints

Different memory sizes, read/write speed and organizations are to be considered depending on the amount and data types of information received by the device.

IEEE standard 1616 defines for each data element: its range, unit of measure, resolution, accuracy, sampling rate, sampling timing and data format. Although this standard does not prescribe which specific data elements shall be recorded, it introduces the reader a list of recommended data elements to be stored.

Depending on the elements selected to be recorded, a minimum memory size will be required.

Given that motor vehicles operate for extensive periods of time, several writing operations are carried out. Consequently, it is imperative that memory modules support virtually unlimited data writings. Therefore, it is not convenient to use electronically erasable memory, such as Electronically Erasable Programmable Read Only Memory (EEPROM) or Flash, because of its limited write endurance. Hence, it is desirable to employ Random Access Memory (RAM) modules instead.

3.3 Communication constraints

Depending on the amount of data to be collected, bandwidth turns into an important decision factor, since network traffic can differ. Thus, potential bottlenecks can be detected early in the development process.

Additionally, the network is to be installed in a hostile environment, so it has to be sufficiently robust and resilient to electromagnetic interference (EMI).

3.4 Processing constraints

A real-time system implies certain time constraints. In this case, each package received by the device may require an amount of processing. Subsequently, the system should have enough Central Processing Unit (CPU) power to process packages and to avoid overflowing its receive buffer.

In addition, the CPU must have an address bus width large enough to be able to address the entire memory locations available to be read or written by the application.

3.5 Domain-specific constraints

The new Argentinean law 24.449 establishes that every particular passenger vehicle must include an event data recorder to be used for accident reconstruction and investigation purposes. However, as this law is rather vague regarding requirements for light motor vehicle EDRs, IEEE regulations are employed as a complement.

4 Proposal of a MVEDR architecture

We introduce a basic architecture for a data acquisition device with SystemC syntax as shown in Figure 1. In the literature we found generic models for distributed data acquisition as show in [10].

4.1 Design

This device contains a CPU, a volatile RAM, a non-volatile memory, an accelerometer and a real time clock. It is connected to a CAN bus and it also provides an interface to retrieve the acquired data.

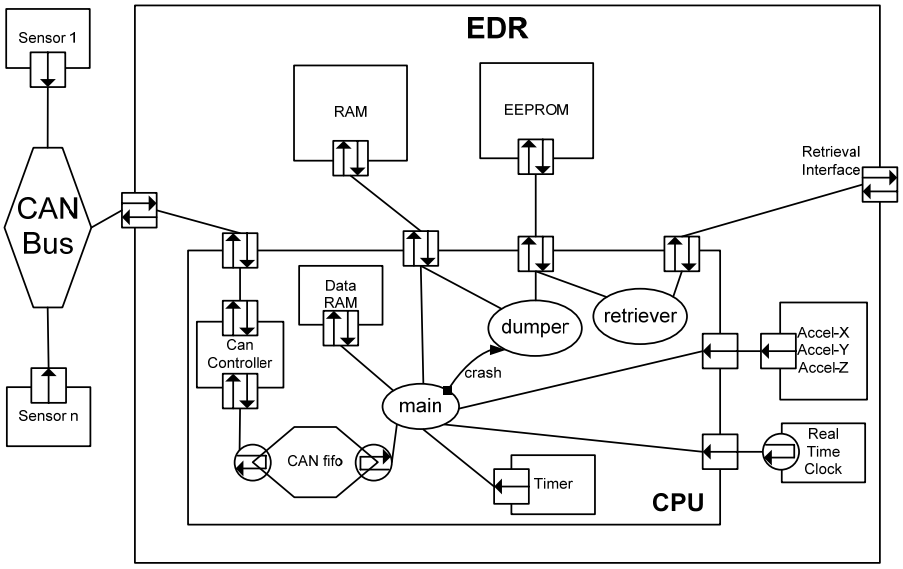

Figure 1: MVEDR Architecture

There is a CAN bus that is described as a Sc_channel where the data acquisition device and sensors are connected to through Sc_ports.

Each Sc_module is assumed to send digital data over the network, i.e., they have built-in analog-to-digital converters that are not described in this model.

The CPU is modeled as a module that contains a data RAM, a network controller and a number of Input/output (I/O) ports to make connections with other components of the architecture. The CPU module behavior is implemented by the Sc_threads and Sc_methods that it comprises, which are responsible for the reception, analysis, selection, processing, storage of the incoming data elements (from the sensors through the network or the I/O ports into the memory), and subsequent retrieval.

The real time clock feeds the system with the current date and time when required by the application. Data elements are time-stamped in order to determine when each one occurred.

Each memory provides read/write access through its Sc_port.

The network controller is embedded within the CPU module to reduce the arithmetic-logic unit load by performing most of the functions related to the networking protocol.

The non-volatile memory is placed in the data acquisition device and is only accessible via the retrieval interface, so that the data stored cannot be manipulated by external agents.

IEEE Standard states that vehicle acceleration can be determined by either air bag accelerometer or other accelerometer. In our design, MVEDR has its own accelerometer because airbag related components are not being modeled.

It is also mandatory to have the exact date and time of the occurrence of each event. The IEEE standard proposes to derive them either from a GPS receiver or from a real time clock. The second option is adopted for simplicity matters.

Besides, the CPU has an internal timer that is used to insert into each data element a timestamp relative to the real time of the event.

4.2 Operation

MVEDR operates as the engine runs. Every sensor in the car samples information and send it through the CAN Bus channel. These packages are received by the Can Controller of the MVEDR and pushed into the CAN FIFO. From there, they are read by the Sc_thread ***main***, which checks whether the sampling rate of the received data element is in conformance with the IEEE Standard or not.

Were to be received more than the expected packages, some of them would be dropped out. The rest is stored in a circular buffer in RAM memory. Each type of element has its own buffer whose size is determined by the equation (1).

$$\text{sampling rate} * \text{sampling timing} * \text{data format} . \quad (1)$$

Sampling rate is the number of samples per second to be recorded, sampling timing is the data recording time and data format is the number of bytes used for representing the element.

In addition, the Sc_thread ***main*** analyzes the information coming from the accelerometer in order to identify the exact moment when an accident occurs. In that case, after the information corresponding to post-crash data is acquired, the Sc_event ***crash*** is triggered and the Sc_thread ***dumper*** dumps the content of RAM memory into the EEPROM.

Finally, the Sc_thread ***retriever*** is awaked whenever the MVEDR is accessed through the retrieval interface. This method accesses the EEPROM thus allowing the download of crash data stored in it.

5 Results

Having the already mentioned model as a reference, we built an implementation in the TLM design language C++/SystemC. As a result, we were able to test, by means of simulations, the feasibility of a device that is compliant with IEEE regulations.

Simulations comprised a car equipped with a MVDER running normally, until an accident was triggered.

After that, the data dumped by the MVEDR to non-volatile memory at the time of the event was retrieved (through the corresponding interface) to verify its conformance with the protocol defined by the IEEE standard.

Simulations provide a great deal of flexibility, thus several instances were run with different configurations with parameterized aspects, such as CPU speed, memory

latencies, CAN network priorities, etc. Naturally, not every instance completed the tests successfully, but this led us to determine the approximate minimum hardware requirements to accomplish the aforementioned standard.

The model presented above has gone through various changes until it reached its final state. Initially, it was a SAM, i.e. untimed. Memory writes were done without any latencies, as well as CPU instructions. In this way, it could be verified, for example, the correct management by the Sc_thread ***main*** of the circular buffer assigned for each data type. In the same way, we could check that the dump from RAM into EEPROM was correctly done by the Sc_thread ***dumper***.

Furthermore, the CAN bus we implemented was able to handle priorities but did not represent the time required to transmit a message.

In the subsequent stages, functional and communicational components were refined so as to make the model behave in a more realistic way regarding an actual hardware implementation. Wait statements were added to the code in order to represent the passage of time. The duration of wait sentences were parameterized so as to be able to instantiate hardware with different properties, such as memory read/write speed.

Our simulations ran with the recommended data elements by the IEEE standard. Given those constraints, the MVEDR had to be capable of processing 3136 data units per second. With this numbers, there are many microcontrollers in the market that can be used as the core of a MVEDR.

6 Conclusions

The adoption of TLM, using a hardware description language like SystemC, provides a highly flexible methodology for development of real time embedded systems. The early executable model, joint with an adequate level of abstraction, allows designers to perform functional verification before hardware implementation is made.

The main advantage acquired in this project from simulations is that they provided early feedback on the performance of the system and allowed to test its functionality under different and diverse conditions.

The results obtained from this project allowed us to foretell a very approximate specification of requirements about hardware components. Besides, the implemented functionality was proven to be in conformance with the IEEE standard.

In a possible next stage, this model can be further refined into a RTL model to be finally deployed into a real hardware implementation.

This development demonstrated the feasibility of an IEEE 1616 compliant MVEDR. Furthermore, it can be the starting point for the development of cost-effective devices to supply the automotive industry with standardized MVEDRs.

7 References

1. Ching-Yao Chan: Trends in Crash Detection and Occupant Restraint Technology. Proceedings of the IEEE, vol. 95, no. 2, pp 388-396 (2007)

2. IEEE: IEEE Standard 1616 – 2004: Standard for Motor Vehicle Event Data Recorders (MVEDRs).

3. C. Shelton, C. Martin: Using Models to Improve the Availability of Automotive Software Architectures. Proceedings of Software Engineering for Automotive Systems. ICSE Workshop, SEAS'07, 20-26 May 2007, pp 9-15 (2007).

4. Cai, L., Gajski, D.: Transaction-level Modeling: an Overview. Proc. of the Int. Conf. on Hardware/Software Codesign and System Synthesis, pp. 19–24. ACM Press, New York (2003)

5. F. Ghenassia: Transaction-level modeling with SystemC. Springer (2005)

6. Shuqing Zhao, D. D. Gajski: Defining an enhanced RTL semantics. Proceedings of Design, Automation and Test in Europe, pp 548-552 (2005).

7. IEEE: IEEE Std. 1666 – 2005: SystemC Language Reference Manual.

8. Open SystemC Initiative, http://www.systemc.org

9. David C. Black, Jack Donovan: SystemC: From the Ground Up. Kluwer Academic Publishers, Boston (2004)

10. J. Ehrlichl, A. Zerroukill, N. Demassieux: Distributed architecture for data acquisition: a generic model. Proc. of the IEEE Instrumentation and Measurement Technology Conference, pp 1180-1185 (1997)